Compatibility via Modernizr

Making web things fit their medium

by **Stu Cox** / **@stucoxmedia**

#McrFRED | 27th June 2013 | Manchester, UK

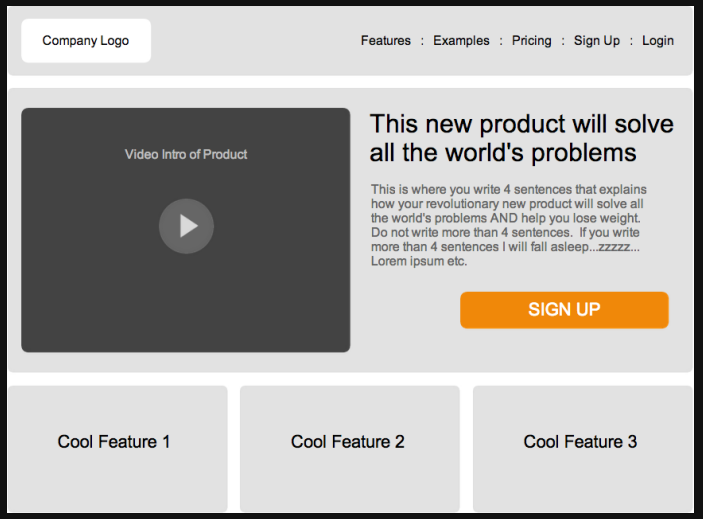
com•pat•i•bil•i•ty |kəmˌpatɪ'bɪlɪti| (abbr.: **compat.**)
*noun (pl. **-i•ties**)*
a state in which two things are able to exist or occur together without problems or conflict.

In this case:

Your web thing ←→ Your user's browser/device

How we did it back in the day

1. Make a **web thing**

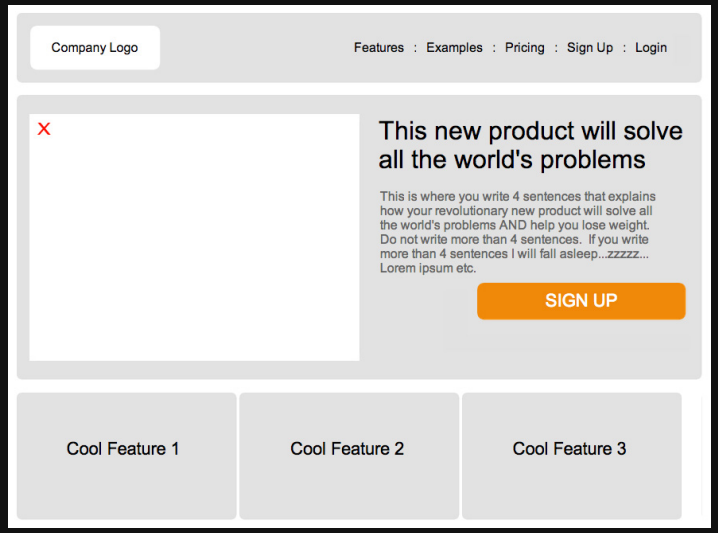


2. Test it in Browser A

3. It works! Hurray!

4. Test it in Browser B

5. Doesn't work :-(



So we hack it.

CSS hacks:

```
.btn {
    *margin-left: -13px;
}
```

User-Agent sniffs:

```
if (navigator.userAgent.match(/MSIE [67]\./)) {
    // Fix for old IE
}
```

These are essentially heuristics.

Heuristics imply assumptions.

- "All browsers which parse CSS hack A also have layout bug B"
- "All browsers which match user-agent C support feature D"
- "I know about every browser my users might use"
- "If my assumptions are true now, they'll also be true in the future"

"I know about every browser my users might use"

- 85 browser versions with > 0.1% market share
- 7,000 different devices login to Facebook every day [1]
- Users have different needs (think accessibility)

[1] techcrunch.com/2012/08/03/vp-mike-schroepfer-7000-different-mobile-devices-access-facebook-every-day/

Browser A

Browser B

Three sources of compatibility problems:

- Features
- Plugins
- Bugs

Features

CSS:

`@font-face`, transitions, animations, flexbox, ...

HTML:

`<audio>`, `<video>`, input types, drag & drop, ...

JavaScript:

History API, IndexedDB, WebSockets, ...

...

Plugins

Platforms/Runtimes:

Flash, Silverlight, Java, ...

Viewers:

PDF, Office documents, ...

...

Bugs

Rendering:

Box model, double margin, ...

Other broken things:

History API in Android 2.x, ...

...

These can all be described under one term:

CAPABILITIES

- Features are capabilities

  "Browser X has the ability to render SVG"
- Plugins *add* capabilities

  "A browser with the Flash plugin has the ability to render Flash media"
- Bugs are incapabilities

  "Browser Y has the ability *NOT* to fuck up the box model"

“The differences between 2 users' browsers can be described (entirely) by the differences between their capability sets.”

Progressive Enhancement

It was going to get mentioned sooner or later.

Providing different experiences for different users, depending on their capabilities.

Think of your web thing as a collection of features.

- Core: the essential bits every user needs
- Enhancements: non-essential additions

The core is smaller than you think.

Each feature depends on capabilities of the browser.

- Core capabilities → "system requirements" for your web thing

  Fewer core capabilities = accessible to more users.
- Enhancement capabilities → tiers of experience

  If a user has the required capabilities, they get the enhancement; otherwise they don't.

What's an enhancement?

Examples:

- Some styling
- Fonts
- Animations
- Audio / video content
- A background image
- A whole functional part – e.g. a chat feature

Atomic Enhancements

- Either applied fully, or not at all
- No side effects when required capabilities aren't present
- *Degrades gracefully*

If enhancements aren't **atomic**, bad things happen.

- Broken layouts
- Javascript errors
- Unusable interfaces

Examples:

```
.module {
    box-sizing: border-box;
    padding: 1em;
    width: 20em;
}
```

If `box-sizing` not supported, layout won't be as expected.

```
// My module
var modules = document.querySelectorAll('.module');
...
```

TypeError: Object #<HTMLDocument> has no method 'querySelectorAll'

How can we ensure enhancements are atomic?

1. Avoid certain dependencies
2. Safety net
3. Feature detect

Feature detection

Testing if the browser offers certain capabilities

Basic pattern

```
if (supportsFeature) {
    // Use feature!
    // Ensure all code depending on this feature is
    // contained here... no side effects!
}
else {
    // Some fallback (optional)
}
```

Techniques

1. ## Does it exist?

```
'geolocation' in navigator
```

2. ## Does it stick?

```
var el = createElement('div');
el.style.cssText = 'filter:blur(2px)';
!!el.style.length   // true if CSS filters supported
```

3. ## Does it work?

```
var image = new Image();
image.onload = function() {
  image.width == 1  // true if WebP supported
}
image.src = '
```

Media Queries

Yep, they're a kind of feature detection too.

In CSS:

```
@media (min-width: 800px) {
    // Has a large viewport
}

@media not (min-width: 800px) {
    // Doesn't have a large viewport
}
```

In JS:

```
if (window.matchMedia('(min-width: 800px)').matches) {
    // Has a large viewport
}
else {
    // Doesn't have a large viewport
}
```

Native Detection

Via `@supports`

In CSS:

```
@supports (display: flex) {
    // Supports flexbox
}

@supports not (display: flex) {
    // Doesn't support flexbox
}
```

In JS:

```
if (window.CSS.supports('display', 'flex') {
    // Supports flexbox
}
else {
    // Doesn't support flexbox
}
```

Fallbacks

- No fallback: "feature gating"
  Efficient, easy to maintain.
- Replace functionality: "polyfilling"
  Can be network and processor intensive, rarely an exact match
- Alternative functionality: "sandwich filling"
  Usually unnecessary…

http://modernizr.com

It's a feature detection library.

Detects for 174+ modern browser capabilities.

Basic patterns

It makes feature detection a breeze

```
if (Modernizr.geolocation) {
    // Use feature!
    // Ensure all code depending on this feature is
    // contained here... no side effects!
}
else {
    // Some fallback (optional)
}
```

```
.geolocation .module {
    /* Styles if geolocation supported */
}
.no-geolocation .module {
    /* Styles if geolocation not supported */
}
```

Custom builds

All killer, no filler

Roll your own

Via `Modernizr.addTest()`

```
Modernizr.addTest('yoda', function () {
    var yoda = document.createElement('yoda');
    return 'theforce' in yoda;
});
```

Conditional Loading

Via `Modernizr.load()`

Avoid heavy loading for browsers which can't use it

```
Modernizr.load({
    test: Modernizr.geolocation,
    yep: 'geo.js',
    nope: 'geo-polyfill.js'
});
```

Rarely need `nope` – big polyfills are a bad idea!

Modernizr v3.0

- New AMD-based internal architecture
- Builds are waaaaay smaller
- 20+ more detects since 2.6.2
- Better handling of async tests
- Uses `@supports` under the hood
- Faster release cycle
- Better documentation
- Easier integration with **grunt-modernizr**

Coming soon, we promise!

See: **Alex Sexton's Modernizr 3 Workflow**

Now I'm going to talk a bit about my involvement in Modernizr.

Undetectables

Unfortunately some things fall under our radar

A good feature detect...

- Gives accurate positives
- Gives accurate negatives
- Is lightweight (fast & small)
- Doesn't make assumptions or use heuristics

Realistically, most detects make some assumptions – we try to minimise these

Rule 1: It must interact with the document

Because that's all we can access from JS.

- ## Styling

  Can't access the pixels on the screen

  Best-guess based on the (re)actions of the DOM

- ## Form UIs

  Appear on top of the document – invisible to us

Rule 2: It shouldn't take any user interaction

- Events

  Can't tell if events (e.g. `DOMContentLoaded`) will be fired at the correct time

- `contenteditable`

We can often give accurate *negatives*, but not positives

Rule 3: Think about older browsers/devices

- ## Using new APIs

  e.g. `window.performance` – tells us nothing about older devices

- ## Touchscreens

  Don't get me started...

You can't detect a touchscreen

Not reliably, anyway

http://stucox.com/blog/you-cant-detect-a-touchscreen/

All techniques either use heuristics or rely on new APIs.

In fact, you can't detect many device features

Not reliably, anyway

Think about the assumptions you're making

Now I'm going to talk a bit about some hand-wavey idealistic stuff.

Managing compatibility

Variation between platforms *is* the web: embrace it.

Consider it from the start of your project.

Idea: modular capability dependencies

RequireJS-like syntax for defining browser dependencies

```
browserRequire(['svg', 'canvas'], function () {

   // Only runs if capabilities available

});
```

Even better: a RequireJS plugin

```
require(['jquery', 'M!svg', 'M!canvas'], function ($) {

   // Only runs if software AND capability dependencies
   // satisfied

});
```

That's all I've got.

www.stucox.com / **@stucoxmedia**